Security – RC4 Example

1 Introduction

Lets consider the stream cipher RC4, but instead of the full 256 bytes, we will use 8 × 3-bits. That is, the state vector **S** is 8 × 3-bits. We will operate on 3-bits of plaintext at a time since **S** can take the values 0 to 7, which can be represented as 3 bits.

Thanks to Thapanee, NazimCSD, Fabian and others for identifying errors in past versions of this example.

2 Example 1

Assume we use a 4 x 3-bit key, **K**, and plaintext **P** as below:

```
K = [1 2 3 6]
P = [1 2 2 2]
```

The first step is to generate the stream.

Initialise the state vector **S** and temporary vector **T**. **S** is initialised so the $\mathbf{S}[i] = i$, and T is initialised so it is the key **K** (repeated as necessary).

```
S = [0 1 2 3 4 5 6 7]
T = [1 2 3 6 1 2 3 6]
```

Now perform the initial permutation on **S**.

```
j = 0;
for i = 0 to 7 do
        j = (j + S[i] + T[i]) mod 8
        Swap(S[i],S[j]);
end
```

We will step through for each iteration of i:

```
For i = 0:
j = (0 + 0 + 1) mod 8
  = 1
Swap(S[0],S[1]);
```

So in the 1st iteration **S**[0] must be swapped with **S**[1] giving:

```
S = [1 0 2 3 4 5 6 7]
```

The results of the remaining 7 iterations are:

```
For i = 1:
j = 3
Swap(S[1],S[3])
S = [1 3 2 0 4 5 6 7];

For i = 2:
j = 0
Swap(S[2],S[0]);
S  = [2 3 1 0 4 5 6 7];

For i = 3:
j = 6;
```

```
Swap(S[3],S[6])
S = [2 3 1 6 4 5 0 7];

For i = 4:
j = 3
Swap(S[4],S[3])
S = [2 3 1 4 6 5 0 7];

For i = 5:
j = 2
Swap(S[5],S[2]);
S  = [2 3 5 4 6 1 0 7];

For i = 6:
j = 5;
Swap(S[6],S[5])
S = [2 3 5 4 6 0 1 7];

For i = 7:
j = 2;
Swap(S[7],S[2])
S = [2 3 7 4 6 0 1 5];
```

Hence, our initial permutation of **S** gives:

```
S= [2 3 7 4 6 0 1 5];
```

Now we generate 3-bits at a time, k, that we XOR with each 3-bits of plaintext to produce the ciphertext. The 3-bits k is generated by:

```
i, j = 0;
while (true) {
        i = (i + 1) mod 8;
        j = (j + S[i]) mod 8;
        Swap (S[i], S[j]);
        t = (S[i] + S[j]) mod 8;
        k = S[t];
}
```

The first iteration:

```
S = [2 3 7 4 6 0 1 5]
i = (0 + 1) mod 8 = 1
j = (0 + S[1]) mod 8 = 3
Swap(S[1],S[3])
S = [2 4 7 3 6 0 1 5]
t = (S[1] + S[3]) mod 8 = 7
k = S[7] = 5
```

Remember, that **P** is:

```
P = [1 2 2 2]
```

So our first 3-bits of ciphertext is obtained by: k XOR P_1

```
5 XOR 1 = 101 XOR 001 = 100 = 4
```

The second iteration:

```
S = [2 4 7 3 6 0 1 5]
i = (1 + 1 ) mod 8 = 2
j = (3 + S[2]) mod 8 = 2
```

```
Swap(S[2],S[2])
S = [2 4 7 3 6 0 1 5]
t = (S[2] + S[2]) mod 8 = 6
k = S[6] = 1
```

Second 3-bits of ciphertext are:

```
1 XOR 2 = 001 XOR 010 = 011 = 3
```

The third iteration:

```
S = [2 4 7 3 6 0 1 5]
i = (2 + 1) mod 8 = 3
j = (2 + S[3]) mod 8 = 5
Swap(S[3],S[5])
S = [2 4 7 0 6 3 1 5]
t = (S[3] + S[5]) mod 8 = 3
k = S[3] = 0
```

Third 3-bits of ciphertext are:

```
0 XOR 2 = 000 XOR 010 = 010 = 2
```

The final iteration:

```
S = [2 4 7 0 6 3 1 5]
i = (1 + 3 ) mod 8 = 4
j = (5 + S[4]) mod 8 = 3
Swap(S[4],S[3])
S = [2 4 7 6 0 3 1 5]
t = (S[4] + S[3]) mod 8 = 6
k = S[6] = 1
```

Last 3-bits of ciphertext are:

```
1 XOR 2 = 001 XOR 010 = 011 = 3
```

So to encrypt the plaintext stream **P** with key **K** with our simplified RC4 stream we get **C**:

```
P = [1 2 2 2]
K = [5 1 0 1]
C = [4 3 2 3]
```

Or in binary:

```
P = 001010010010
K = 101001000001
C = 100011010011
```